A Tree-Based Algorithm for Ranking Web Services

Fatma Ezzahra Gmati[1], Nadia Yacoubi-Ayadi[1], Afef Bahri[2], Salem Chakhar[3,4] and Alessio Ishizaka[3,4]
[1]*RIADI Research Laboratory, National School of Computer Sciences, University of Manouba, Manouba, Tunisia*
[2]*MIRACL Laboratory, High School of Computing and Multimedia, University of Sfax, Sfax, Tunisia*
[3]*Portsmouth Business School, University of Portsmouth, Portsmouth, UK*
[4]*Centre for Operational Research and Logistics, University of Portsmouth, Portsmouth, UK*
{*fatma.ezzahra.gmati, nadia.yacoubi.ayadi, afef.bahri*}*@gmail.com,* {*salem.chakhar, alessio.ishizaka*}*@port.ac.uk*

Keywords: Web service, Service scoring, Service ranking, Tree structure, Algorithm

Abstract: The aim of this paper is to propose a new algorithm for Web services ranking. The proposed algorithm relies on a tree data structure that is constructed based on the scores of Web services. Two types of scores are considered, which are computed by respectively selecting the edge with the minimum or the edge with the maximum weight in the matching graph. The construction of the tree requires the successive use of both scores, leading to two different versions of the tree. The final ranking is obtained by applying a pre-order traversal on the tree and picks out all leaf nodes ordered from the left to the right. The performance evaluation shows that the proposed algorithm is most often better than similar ones.

1 INTRODUCTION

Although the semantic matchmaking (Paolucci et al., 2002; Doshi et al., 2004; Bellur and Kulkarni, 2007; Fu et al., 2009; Chakhar, 2013; Chakhar et al., 2014; Chakhar et al., 2015) permits to avoid the problem of simple syntactic and strict capability-based matchmaking, it is not very suitable for efficient Web service selection. This is because it is difficult to distinguish between a pool of similar Web services (Rong et al., 2009).

A possible solution to this issue is to use some appropriate techniques and some additional information to rank the Web services delivered by the semantic matching algorithm and then provide a manageable set of 'best' Web services to the user from which s/he can select one Web service to deploy. Several approaches have been proposed to implement this idea (Manikrao and Prabhakar, 2005; Maamar et al., 2005; Kuck and Gnasa, 2007; Gmati et al., 2014; Gmati et al., 2015).

The objective of this paper is to introduce a new Web services ranking algorithm. The proposed algorithm relies on a tree data structure that is constructed based on the scores of Web services. Two types of scores are considered, which are computed by respectively selecting the edge with the minimum or the edge with the maximum weight in the matching graph. The construction of the tree requires the successive use of both scores, leading to two different versions of the tree. The final ranking is obtained by applying a pre-order traversal on the tree and picks out all the leaf nodes ordered from the left to the right.

The rest of the paper is organized as follows. Section 2 provides a brief review of Web services matching. Section 3 shows how the similarity degrees are computed. Section 4 presents the Web services scoring technique. Section 5 details the tree-based ranking algorithm. Section 6 evaluates the performance of the proposed algorithm. Section 7 discusses some related work. Section 8 concludes the paper.

2 MATCHING WEB SERVICES

The main input for the ranking algorithm is a set of Web services satisfying the user requirements. In this section, we briefly review three matching algorithms that can be used to identify this set of Web services. These algorithms support different levels of customization. This classification of matching algorithms according to the levels of customization that they support enrich and generalize our previous work in (Chakhar, 2013; Chakhar et al., 2014; Gmati et al., 2014; Gmati et al., 2014).

In the rest of this paper, a Web service S is defined as a collection of attributes that describe the service.

Let $S.A$ denotes the set of attributes of service S and $S.A_i$ denotes each member of this set. Let $S.N$ denotes the cardinality of this set.

2.1 Trivial Matching

The trivial matching assumes that the user can specify only the functional specifications of the desired service. Let S^R be the Web service that is requested, and S^A be the Web service that is advertised. A sufficient trivial match exists between S^R and S^A if for *every* attribute in $S^R.A$ there exists an identical attribute of $S^A.A$ and the similarity between the values of the attributes does not fail. Formally,

$$\begin{array}{l}\forall_i \exists_j (S^R.A_i = S^A.A_j) \wedge \mu(S^R.A_i = S^A.A_j) \succ \textbf{Fail} \\ \Rightarrow \text{SuffTrivialMatch}(S^R, S^A) \quad 1 \leq i \leq S^R.N.\end{array} \tag{1}$$

where: $\mu(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity degree between the requested Web service and the advertised Web service on the jth attribute A_j. According to this definition, all the attributes of the requested service S^R should be considered during the matching process. This is the default case with no support of customization.

The trivial matching is formalized in Algorithm 1. This algorithm follows directly from Sentence (1).

Algorithm 1: Trivial Matching

```
Input  : S^R, // Requested service.
         S^A, // Advertised service.
Output: Boolean, // fail/success.
1  while (i ≤ S^R.N) do
2      Append S^R.A_i to rAttrSet;
3      while (k ≤ S^A.N) do
4          if (S^A.A_k = S^R.A_i) then
5              Append S^A.A_k to aAttrSet;
6          k ⟵ k + 1;
7      i ⟵ i + 1;
8  while (t ≤ S^R.N) do
9      if (μ(rAttrSet[t], aAttrSet[t]) = Fail) then
10         return fail;
11     t ⟵ t + 1;
12 return success;
```

The definition of the semantic degrees used in Sentence (1) and Algorithm 1 (and also in the two other matching algorithms presented later) is given in Section 3.

2.2 Partially Parameterized Matching

In this case, the user can specify the list of attributes to consider during the matching process. In order to allow this, we use the concept of Attributes List that serves as a parameter to the matching process. An Attributes List, L, is a relation consisting of one attribute, $L.A$ that describes the service attributes to be compared. Let $L.A_i$ denotes the service attribute value in the ith tuple of the relation. $L.N$ denotes the total number of tuples in L. We assume that the order of attributes in L is randomly specified.

Let S^R be the service that is requested, S^A be the service that is advertised, and L be an Attributes List. A sufficient partially parameterized match exists between S^R and S^A if for *every* attribute in $L.A$ there exists an identical attribute of S^R and S^A and the similarity between the values of the attributes does not fail. Formally,

$$\begin{array}{l}\forall_i \exists_{j,k} (L.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succ \textbf{Fail} \\ \Rightarrow \text{SuffPartiallyParamMatch}(S^R, S^A) \quad 1 \leq i \leq L.N.\end{array} \tag{2}$$

According to this definition, only the attributes specified by the user in the Attributes List are considered during the matching process.

The partially parameterized matching is formalized in Algorithm 2 that follows directly from Sentence (2).

Algorithm 2: Partially Parameterized Matching

```
Input  : S^R, // Requested service.
         S^A, // Advertised service.
         L, // Attributes List.
Output: Boolean, // fail/success.
1  while (i ≤ L.N) do
2      while (j ≤ S^R.N) do
3          if (S^R.A_j = L.A_i) then
4              Append S^R.A_j to rAttrSet;
5          j ⟵ j + 1;
6      while (k ≤ S^A.N) do
7          if (S^A.A_k = L.A_i) then
8              Append S^A.A_k to aAttrSet;
9          k ⟵ k + 1;
10     i ⟵ i + 1;
11 while (t ≤ L.N) do
12     if (μ(rAttrSet[t], aAttrSet[t]) = Fail) then
13         return fail;
14     t ⟵ t + 1;
15 return success;
```

2.3 Fully Parameterized Matching

The fully parameterized matching supports three customizations by allowing the user to specify: (i) the list of attributes to be considered; (ii) the order in which these attributes are considered; and (iii) a desired similarity degree for each attribute. In order to support all the above-cited customizations, we used the concept of Criteria Table, introduced by (Doshi et al., 2004).

A Criteria Table, C, is a relation consisting of two attributes, $C.A$ and $C.M$. $C.A$ describes the service attribute to be compared, and $C.M$ gives the *least* preferred similarity degree for that attribute. Let $C.A_i$ and $C.M_i$ denote the service attribute value and the desired degree in the ith tuple of the relation. $C.N$ denotes the total number of tuples in C.

Let S^R be the service that is requested, S^A be the service that is advertised, and C be a Criteria Table. A sufficient fully parameterized match exists between S^R and S^A if for *every* attribute in $C.A$ there exists an identical attribute of S^R and S^A and the values of the attributes satisfy the desired similarity degree as specified in $C.M$. Formally,

$$\begin{aligned}&\forall_i \exists_{j,k} (C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i \\ &\Rightarrow \text{SuffFullyParamMatch}(S^R, S^A) \quad 1 \leq i \leq C.N.\end{aligned} \tag{3}$$

According to this definition, only the attributes specified by the user in the Criteria Table C are considered during the matching process. The fully parameterized matching is formalized in Algorithm 3 that follows directly from Sentence (3).

Algorithm 3: Fully Parameterized Matching

```
Input  : S^R, // Requested service.
         S^A, // Advertised service.
         C, // Criteria Table.
Output: Boolean, // fail/success.
1  while (i ≤ C.N) do
2      while (j ≤ S^R.N) do
3          if (S^R.A_j = C.A_i) then
4              Append S^R.A_j to rAttrSet;
5          j ⟵ j + 1;
6      while (k ≤ S^A.N) do
7          if (S^A.A_k = C.A_i) then
8              Append S^A.A_k to aAttrSet;
9          k ⟵ k + 1;
10     i ⟵ i + 1;
11 while (t ≤ C.N) do
12     if (μ(rAttrSet[t], aAttrSet[t]) ≺ C.M_t) then
13         return fail;
14     t ⟵ t + 1;
15 return success;
```

The complexity of the matching algorithms are detailed in (Gmati, 2015).

3 SIMILARITY DEGREE COMPUTING APPROACH

In this section, we first define the similarity degree used in the matching algorithms and then discuss how it is computed.

3.1 Similarity Degree Definition

A semantic match between two entities frequently involves a similarity degree that quantifies the semantic distance between the two entities participating in the match. The similarity degree, $\mu(\cdot,\cdot)$, of two service attributes is a mapping that measures the semantic distance between the conceptual annotations associated with the service attributes. Mathematically,

$$\mu : A \times A \rightarrow \quad \{\text{Exact, Plug-in, Subsumption, Container, Part-of, Fail}\}$$

where A is the set of all possible attributes. The definitions of the mapping between two conceptual annotations are given in (Doshi et al., 2004; Chakhar, 2013; Chakhar et al., 2014).

A preferential total order is established on the above mentioned similarity maps: Exact $\succ$ Plug-in $\succ$ Subsumption $\succ$ Container $\succ$ Part-of $\succ$ Fail; where $a \succ b$ means that a is preferred over b.

3.2 Similarity Degree Computing

To compute the similarity degrees, we generalized and implemented an idea proposed by (Bellur and Kulkarni, 2007). This idea starts by constructing a bipartite graph where the vertices in the left side correspond to the concepts associated with advertised services, while those in the right side correspond to the concepts associated with the requested service. The edges correspond to the semantic relationships between concepts. The authors in (Bellur and Kulkarni, 2007) assign a weight to each edge and then apply the Hungarian algorithm (Kuhn, 1955) to identify the complete matching that minimizes the maximum weight in the graph. The final returned degree is the one corresponding to the maximum weight in the graph.

We generalize this idea as follows. Let first assume that the output of the matching algorithm is a list mServices of matching Web services. The generic structure of a row in mServices is as follows:

$$(S_i^A, \mu(S_i^A.A_1, S^R.A_1), \cdots, \mu(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A is an advertised Web service, S^R is the requested Web service, N the total number of attributes and $\mu(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity degree between the requested Web service and the advertised Web service on the jth attribute A_j.

Two versions can be distinguished for the definition of the list mServices, along with the way the similarity degrees are computed. The first version of mServices is as follows:

$$(S_i^A, \mu_{\max}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\max}(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A, S^R and N are as defined above; and $\mu_{\max}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity degree between the requested Web service and the advertised Web service on the jth attribute A_j computed by selecting the edge with the **maximum weight** in the matching graph.

The second version of mServices is as follows:

$$(S_i^A, \mu_{\min}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\min}(S_i^A.A_N, S^R.A_N)),$$

where S_i^A, S^R and N are as defined above; and $\mu_{\min}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity degree between the requested Web service and the advertised Web service on the jth attribute A_j computed by selecting the edge with the **minimum weight** in the matching graph.

4 SCORING WEB SERVICES

In this section, we propose a technique to compute the scores of the Web services based on the input data.

4.1 Score Definition

First, we need to assign a numerical weight to every similarity degree as follows: Fail: w_1, Part-of: w_2, Container: w_3, Subsumption: w_4, Plug-in: w_5 and Exact: w_6. These degrees correspond to the possible values of $\mu_\diamond(S_i^A.A_j, S^R.A_j)$ with $(j = 1, \cdots, N)$ and where S_i^A, S^R and N are as defined above; and $\diamond \in \{\min, \max\}$. In this paper, we assume that the weights are computed as follows:

$$w_1 \geq 0, \quad (4)$$

$$w_i = (w_{i-1} \cdot N) + 1, \quad i = 2, \cdots, N; \quad (5)$$

where N is the number of attributes. This way of weights computation ensures that a single higher similarity degree will be greater than a set of N similarity degrees of lower weights taken together. Indeed, the weights values verify the following condition:

$$w_i > w_j \cdot N, \quad \forall i > j. \quad (6)$$

Then, the initial score of an advertised Web service S^A is computed as followers:

$$\rho_\diamond(S^A) = \sum_{i=1}^{i=N} w_i. \quad (7)$$

The scores as computed by Equation (7) are not in the range 0-1. Hence, we need to normalize these scores by using the following procedure:

$$\rho'_\diamond(S^A) = \frac{\rho_\diamond(S^A) - \min_K \rho_\diamond(S^K)}{\max_K \rho_\diamond(S^K) - \min_K \rho_\diamond(S^K)}. \quad (8)$$

This normalization procedure assigns to each advertised Web service S^A the percentage of the extent of the similarity degrees scale (i.e., $\max_K \rho_\diamond(S^K) - \min_K \rho_\diamond(S^K)$). It ensures that the scores cover all the range [0,1]. In other words, the lowest score will be equal to 0 and the highest score will be equal to 1. We note that other normalization procedures can be used (see (Gmati, 2015) for more details on these procedures).

4.2 Score Computing Algorithm

The computing of the normalized scores is operationalized by Algorithm 4. This algorithm takes as input a list mServices of Web services each is described by a set of N similarity degrees where N is the number of attributes. The data structure mServices used as input assumed to be defined as:

$$(S_i^A, \mu_\diamond(S_i^A.A_1, S^R.A_1), \cdots, \mu_\diamond(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A is an advertised Web service, S^R is the requested Web service, N the total number of attributes; $\diamond \in \{\min, \max\}$, and $\mu_\diamond(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity degree between the requested Web service and the advertised Web service on the jth attribute A_j computed using one of the of two versions given in Section 3.2.

Algorithm 4: ComputeNormScores

```
Input  : mServices , // List of Web services.
         N, // Number of attributes.
Output: mServices , // List of Web services with normalized scores.
1  r ⟵ length (mServices );
2  t ⟵ 1;
3  while (t ≤ r) do
4      simDegrees⟵ the tth row in mServices ;
5      s ⟵ ComputeInitialScore(simDegrees,N, w);
6      mServices [t,N+2] ⟵ s;
7  a ⟵ mServices [1,N+2];
8  b ⟵ mServices [1,N+2];
9  t ⟵ 1;
10 while (t < r) do
11     if (a > mServices[t+1,N+2])) then
12         a ⟵ mServices [t+1,N+2];
13     if (b < mServices[t+1,N+2])) then
14         b ⟵ mServices [t+1,N+2];
15 t ⟵ 1;
16 while (t ≤ r) do
17     ns ⟵ (mServices [t,N+2] − a)/(b − a);
18     mServices [t,N+2] ⟵ ns;
19 return mServices ;
```

At the output, Algorithm 4 provides an updated version of mServices by adding to it the normalized scores of the Web services:

$$(S_i^A, \mu_\diamond(S_i^A.A_1, S^R.A_1), \cdots, \mu_\diamond(S_i^A.A_N, S^R.A_N), \rho'_\diamond(S_i^A)),$$

where $\rho'_{\diamond}(S_i^A)$ is the normalized score of Web service S_i^A.

The function ComputeInitialScore in Algorithm 4 permits to compute the initial scores of Web services using Equation (7). Function ComputeInitialScore takes a row simDegrees of similarity degrees for a given Web service and computes the initial score of this Web service based on Equation (7). The list simDegrees is assumed to be defined as follows:

$$(S_i^A, \mu_{\diamond}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\diamond}(S_i^A.A_N, S^R.A_N)),$$

where S_i^A, S^R, N and $\mu_{\diamond}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ are as defined previously. It is easy to see that the list simDegrees is a row from the data structure mServices introduced earlier.

The complexity of the score computing algorithm is detailed in (Gmati, 2015).

5 TREE-BASED RANKING OF WEB SERVICES

In this section, we propose a new algorithm for Web services ranking.

5.1 Principle

The basic idea of the proposed ranking algorithm is to use the two types of scores introduced in Section 3.2 to first construct a tree T and then apply a *pre-order tree traversal* on T to identify the final and best ranking. The construction of the tree requires thus to use the score computing algorithm twice, once using the minimum weight value and once using a the maximum weight value (see Section 3.2). In what follows, we assume that the input of the ranking algorithm is a list of matching Web services mServScores defined as follows:

$$(S_i^A, \rho'_{\min}(S_i^A), \rho'_{\max}(S_i^A)),$$

where: S_i^A is an advertised Web service; and $\rho'_{\min}(S_i^A)$ and $\rho'_{\max}(S_i^A)$ are the normalized scores of S_i^A that are computed based on the minimum weight value and the maximum weight value, respectively.

The tree-based ranking algorithm given later in Section 5.3 is composed of two main phases. The objective of the first phase is to construct the tree T. The objective of the second phase is to identify the final ranking using a pre-order tree traversal on T.

5.2 Tree Construction

The tree construction process starts by defining a root node containing the initial list of Web services and then uses different node splitting functions to progressively split the nodes of each level into a collection of new nodes. The tree construction process is designed such that each of the leaf nodes will contain a single Web service. Let first introduce the node splitting functions.

5.2.1 Node Splitting Functions

The first node splitting function is formalized in Algorithm 5. This function receives in entry one node with a list of Web services and generates a ranked list of nodes each with one or more Web services. The function SortScoresMax and SortScoresMin in Algorithm 5 permit to sort Web services based either on the maximum edge value or the minimum edge value, respectively. Function Split permit to split the Web services in L_0 into a set of sublists, each with a subset of services having the same score. The instructions in lines 8-11 in Algorithm 5 permit to create a node for each sublist in Sublists. The algorithm outputs a list T of nodes ordered, according to the scores of Web services in each node, from the left to the right.

Algorithm 5: Node splitting

```
   Input  : Node, // A node.
            NodeSplittingType , // Node splitting type.
   Output: L, // List of ranked nodes.
 1 L ← ∅;
 2 L0 ← Node.mServScores;
 3 if (NodeSplittingType = 'Max') then
 4     L0 ← SortScoresMax (L0);
 5 else
 6     L0 ← SortScoresMin (L0);
 7 SubLists ← Split (L0);
 8 for (for each elem in SubLists) do
 9     create node n;
10     n.add(elem);
11     L.add(n);
12 return L;
```

The second node splitting function is formalized in Algorithm 6. This functions permits to split randomly a node into a set of nodes, each with one Web service.

5.2.2 Tree Construction Algorithm

The tree construction process contains four steps:

1. *Construction of Level 0.* In this initialization step, we create a root node r containing the list of Web services to rank. At this level, the tree T contains only the root node.

2. *Construction of Level 1.* Here, we split the root node into a set of nodes using the node splitting

function (Algorithm 5) with the desired sorting type (i.e. based either on the maximum edge value or the minimum edge value). At the end of this step, a new level is added to the tree T. The nodes of this level are ordered from left to right. Each node of this level will contain one or several Web services. The Web services of a given node will have the same score.

3. *Construction of Level 2*. Here, we split the nodes of the previous level that have more than one Web services using the node splitting function (Algorithm 5) with a different sorting type than the previous step. At the end of this step, a new level is added to the tree T. The nodes of this level are ordered from left to right. Each node of this level will contain one or several Web services. The Web services of a given node will have the same score.

4. *Construction of Level 3*. Here, we apply the random splitting function given in Algorithm 6 to randomly split the nodes of the previous level that has more than one Web service. At the end of this step, a new level is added to the tree T. All the nodes of this level contain only one Web service.

Algorithm 6: Random Node Splitting

```
   Input  : Node, // A node.
   Output: L, // List of ranked nodes.
1  L ← ∅;
2  L_0 ← Node.mServScores;
3  i ← |L|;
4  Z ← L_0;
5  while i > 0 do
6      select randomly a service n from Z;
7      create node n ;
8      n.add(elem);
9      L.add(n);
10     i ← i − 1;
11     Z ← Z \ {n};
12 return L;
```

Figure 1 provides an example of a tree constructed based on this idea.

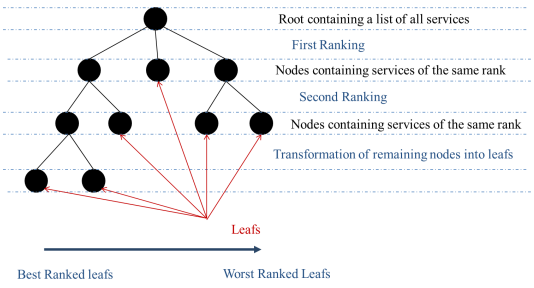


Figure 1: Tree structure.

We may distinguish two version for the tree construction process. The first version, which is shown in Algorithm 7, uses the node splitting based on the Max to generate the nodes of the second level and then the node splitting based on the Min to generate the nodes of the third level. The second version, which is not give in this paper, uses an opposite order.

Algorithm 7: Tree Construction (Max-Min)

```
   Input  : mServScores, // Initial list of services with their scores.
   Output: T, // Tree.
1  create node Node;
2  Node.add(mServScores);
3  add Node as root of the tree T;
4  FirstLevelNodes ← NodeSplitting(Node,'Max');
5  for each node n_1 in FirstLevelNodes do
6      add n_1 as a child of the root;
7      if n_1 is not a leaf then
8          SecondLevelNodes ← NodeSplitting(Node,'Min');
9          for each node n_2 in SecondLevelNodes do
10             add n_2 as a child of n_1;
11             if n_2 is not a leaf then
12                 ThirdLevelNodes ←
                   RandomNodeSplitting(n_2);
13                 for each node n_3 in ThirdLevelNodes do
14                     add n_3 as a child of n_2;
15 return T;
```

5.3 Tree-Based Ranking Algorithm

We propose here a new algorithm for implementing the solution proposed in Section 5.1. The idea of the algorithm is to construct first a tree T using one of the algorithms discussed in the previous section and then scanning through this tree in order to identify the final ranking. The identification of the best and final ranking needs to apply a *tree traversal* (also known as *tree search*) on the tree T. The tree traversal refers to the process of visiting each node in a tree data structure, exactly once, in a systematic way. Different types of traversals are possible: pre-order, in-order and post-order. They differ by the order in which the nodes are visited. In this paper, we will use the pre-order type.

The idea discussed in the previous paragraph is implemented by Algorithm 8. The main input for this algorithm is the initial list mServScores of Web services with their scores. This list is assumed to have the same structure as indicated in Section 5.1. The output of Algorithm 8 is a list FinalRanking of ranked Web services.

The functions ComputeNormScoresMax and ComputeNormScoresMin are not given in this paper. They are similar to function ComputeNormScores introduced previously in Algorithm 4. The scores in function ComputeNormScoresMax are computed

by selecting the edge with the **maximum weight** in the matching graph while the scores in function ComputeNormScoresMin are computed by selecting the edge with the **minimum weight** in the matching graph.

Algorithm 8: Tree-Based Ranking

```
Input  : mServScores, // List of matching services with their scores.
         N, // Number of attributes.
         SplittingOrder , // Nodes splitting order.
Output: FinalRanking , // Ranked list of Web services.
1 T ← ∅;
2 mServScores ← ComputeNormScoresMax( mServScores,N);
3 mServScores ← ComputeNormScoresMin( mServScores,N);
4 if (SplittingOrder = 'MaxMin') then
5     T ← ConstructTreeMaxMin( mServScores);
6 else
7     T ← ConstructTreeMinMax( mServScores);
8 FinalRanking ← TreeTraversal (T);
9 return FinalRanking ;
```

Algorithm 8 can be organized into two phases. The first phase concerns the construction of the tree T. This phase is implemented by the instructions in lines 1-7. According to the type of nodes splitting order (Max-Min or Min-Max), Algorithm 8 uses either function ConstructTreeMaxMin (for Max-Min order) or function ConstructTreeMinMax (for Min-Max order) to construe the tree.

The second phase of Algorithm 8 concerns the identification of the best and final ranking by applying a pre-order tree traversal on the tree T. The pre-order tree traversal contains three main steps: (1) examine the root element (or current element); (2) traverse the left subtree by recursively calling the pre-order function; and (3) traverse the right subtree by recursively calling the pre-order function. The pre-order tree traversal is implemented by Algorithm 9.

Algorithm 9: Tree Traversal

```
Input  : T, // Tree.
Output: L, //Final ranking.
1 L ← ∅;
2 CurrNode ← T.R;
3 if CurrNode contains a single Web service then
4     Let currService be the single Web service in CurrNode ;
5     L.Append(currService );
6 for each child f of CurrNode do
7     Traverse (f, CurrNode);
8 return L;
```

Algorithm 9 takes as input a tree T and generates the final ranking list L. Algorithm 9 scans the tree T and picks out all leaf nodes of T ordered from the left to the right.

The complexity of the tree-based ranking algorithms are detailed in (Gmati, 2015).

6 PERFORMANCE EVALUATION

In what follows, we first compare the tree-based algorithm presented in this paper to the score-based (Gmati et al., 2014) and rule-based (Gmati et al., 2015) ranking algorithms and then discuss the effect of the edge weight order on the final results.

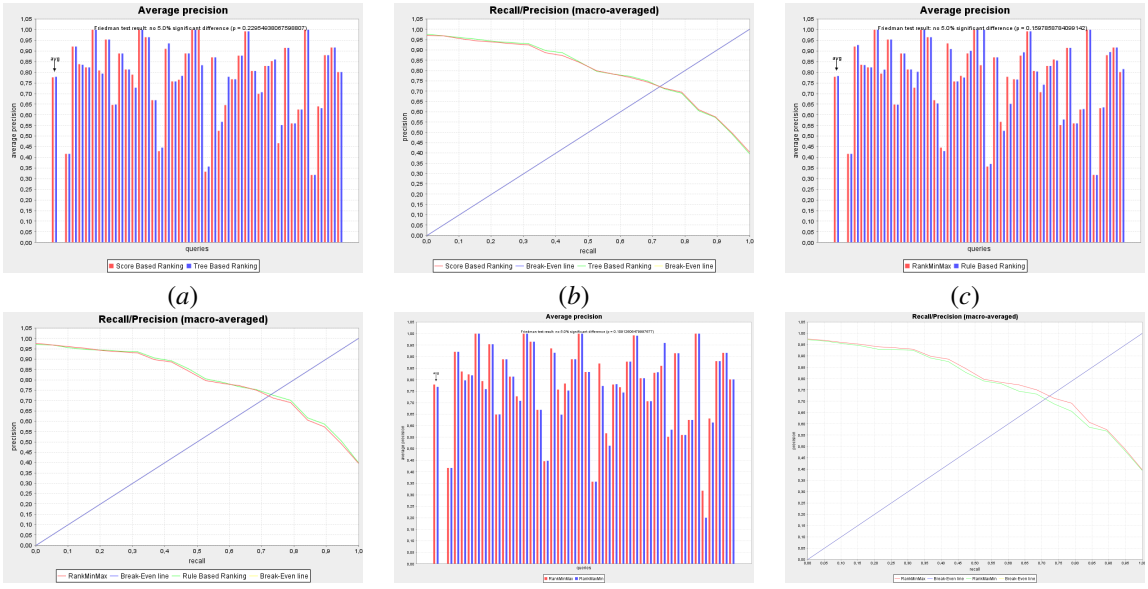
The SME2 (Klusch et al., 2010; Küster and König-Ries, 2010), which is an open source tool for testing different semantic matchmakers in a consistent way, is used for this comparative study. The SME2 uses OWLS-TC collections to provide the matchmakers with Web service descriptions, and to compare their answers to the relevance sets of the various queries.

The experimentations have been conducted on a Dell Inspiron 15 3735 Laptop with an Intel Core I5 processor (1.6 GHz) and 2 GB of memory. The test collection used is OWLS-TC4, which consists of 1083 Web service offers described in OWL-S 1.1 and 42 queries.

Figures 2.*a* and 2.*b* show that the tree-based ranking algorithm has better average precision and recall precision than score-based ranking algorithm. Figure 2.*c* shows that the tree-based ranking algorithm has a slightly better average precision than rule-based ranking algorithm. Figure 2.*d* shows that rule-based ranking algorithm is slightly faster than tree-based ranking algorithm.

The tree-based ranking algorithm is designed to work with either the Min-Max or Max-Min versions of the tree construction algorithm. As discussed in Section 5.2.2, the main difference between these versions is the order in which the edge weight values are used. To study the effect of tree construction versions on the final results, we conduced a series of experiments using the OWL-TC test collection. We evaluated the two versions in respect to the Average Precision and the Recall/Precision metrics.

The result of the comparison is shown in Figures 2.*e* and 2.*f*. According to these figures, we conclude that Min-Max version outperforms the Max-Min. However, this final constatation should not be taken as a rule since it might depend on the considered test collection.



(*a*) (*b*) (*c*)

(*d*) (*e*) (*f*)

Figure 2: Performance analysis.

7 RELATED WORK

We may distinguish three basic Web services ranking approaches, along with the nature of information used for the ranking process: (i) ranking approaches based only on the Web service description information; (ii) ranking approaches based on external information; and (iii) ranking approaches based on the user preferences.

The first ranking approach relies only on the information available in the Web service description, which concerns generally the service capability (IOPE attributes), the service quality (QoS attributes) and/or the service property (additional information). This type of ranking is the most used, since all the needed data is directly available in the Web service description. Among the methods based on this approach, we cite (Manoharan et al., 2011) where the authors combine the QoS and the fuzzy logic and propose a ranking matrix. However, this approach is centered only on the QoS and discards the other Web service attributes. We also mention the work of (Skoutas et al., 2010) where the authors propose a ranking method that computes a dominance score between services. The calculation of these scores requires a pairwise comparison that increases the time complexity of the ranking algorithm.

The second ranking approach uses both the Web service description and other external information (see, e.g., (Kuck and Gnasa, 2007)(Kokash et al., 2007)(Maamar et al., 2005)(Manikrao and Prabhakar, 2005)(Segev and Toch, 2011)). For instance, the authors in (Kuck and Gnasa, 2007) take into account additional information concerning time, place and location in order to rank Web services. In (Kokash et al., 2007), the authors rank Web services on the basis of the user past behavior. The authors in (Maamar et al., 2005) rely their ranking on the customer and providers situations. However, these additional constraints make the system more complex. The authors in (Segev and Toch, 2011) extract the context of Web services and employ it as an additional information during the ranking. The general problem with this type of approaches is that the use of external information can only be performed in some situations where the data is available, which is not always the case in practice.

The third ranking approach is based on the user preference. In (Beck and Freitag, 2006), for instance, the authors use some constraints specified by the user. A priority is then assigned to each constraint or group of constraints. The algorithm proposed by (Beck and Freitag, 2006) uses then a tree structure to perform the matching and ranking procedure.

8 CONCLUSION

In this paper, we proposed a new algorithm for Web services ranking. This algorithm relies on a tree data structure that is constructed based on two types of Web services scores. The final ranking is obtained by applying a pre-order traversal on the tree and picks

out all leaf nodes ordered from the left to the right. The tree-based algorithm proposed in this paper is compared two other ones: score-based algorithm and rule-based algorithm. The performance evaluation of the three algorithms shows that the tree-based algorithm outranks the score-based algorithm in all cases and most often better than the rule-based algorithm.

REFERENCES

Beck, M. and Freitag, B. (2006). Semantic matchmaking using ranked instance retrieval. In *Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval*, Seoul, South Korea.

Bellur, U. and Kulkarni, R. (2007). Improved matchmaking algorithm for semantic Web services based on bipartite graph matching. In *IEEE International Conference on Web Services*, pages 86–93, Salt Lake City, Utah, USA.

Chakhar, S. (2013). Parameterized attribute and service levels semantic matchmaking framework for service composition. In *Fifth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2013)*, pages 159–165, Seville, Spain.

Chakhar, S., Ishizaka, A., and Labib, A. (2014). Qos-aware parameterized semantic matchmaking framework for Web service composition. In Monfort, V. and Krempels, K.-H., editors, *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 1, Barcelona, Spain, 3-5 April, 2014*, pages 50–61, Barcelona, Spain. SciTePress.

Chakhar, S., Ishizaka, A., and Labib, A. (2015). Semantic matching-based selection and qos-aware classification of web services. In *Selected Papers from The 10th International Conference on Web Information Systems and Technologies (WEBIST 2014)*, Lecture Notes in Business Information Processing. Springer-Verlag, Berlin, Heidelberg, Germany. forthcoming.

Doshi, P., Goodwin, R., Akkiraju, R., and Roeder, S. (2004). Parameterized semantic matchmaking for workflow composition. IBM Research Report RC23133, IBM Research Division.

Fu, P., Liu, S., Yang, H., and Gu, L. (2009). Matching algorithm of Web services based on semantic distance. In *International Workshop on Information Security and Application (IWISA 2009)*, pages 465–468, Qingdao, China.

Gmati, F. E. (2015). Parameterized semantic matchmaking and ranking framework for web service composition. Master thesis, Higher Business School of Tunis (University of Manouba) and Institute of Advanced Business Studies of Carthage (University of Carthage), Tunisia.

Gmati, F. E., Yacoubi-Ayadi, N., Bahri, A., Chakhar, S., and Ishizaka, A. (2015). *PMRF: Parameterized matching-ranking framework*. Studies in Computational Intelligence. Springer. forthcoming.

Gmati, F. E., Yacoubi-Ayadi, N., and Chakhar, S. (2014). Parameterized algorithms for matching and ranking Web services. In *Proceedings of the On the Move to Meaningful Internet Systems: OTM 2014 Conferences 2014*, volume 8841 of *Lecture Notes in Computer Science*, pages 784–791. Springer.

Klusch, M., Dudev, M., Misutka, J., Kapahnke, P., and Vasileski, M. (2010). *SME^2 Version 2.2. User Manual*. The German Research Center for Artificial Intelligence (DFKI), Germany.

Kokash, N., Birukou, A., and DAndrea, V. (2007). Web service discovery based on past user experience. In *Proceedings of the 10th International Conference on Business Information Systems*, pages 95–107.

Kuck, J. and Gnasa, M. (2007). Context-sensitive service discovery meets information retrieval. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom'07)*, pages 601–605.

Kuhn, H. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97.

Küster, U. and König-Ries, B. (2010). Measures for benchmarking semantic Web service matchmaking correctness. In *Proceedings of the 7th International Conference on The Semantic Web: Research and Applications - Volume Part II*, ESWC'10, pages 45–59, Berlin, Heidelberg. Springer-Verlag.

Maamar, Z., Mostefaoui, S., and Mahmoud, Q. (2005). Context for personalized Web services. In *In Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, pages 166b–166b.

Manikrao, U. and Prabhakar, T. (2005). Dynamic selection of Web services with recommendation system. In *Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP 2005)*, pages 117–121.

Manoharan, R., Archana, A., and Cowla, S. (2011). Hybrid Web services ranking algorithm. *International Journal of Computer Science Issues*, 8(2):83–97.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. (2002). Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 333–347, London, UK, UK. Springer-Verlag.

Rong, W., Liu, K., and Liang, L. (2009). Personalized Web service ranking via user group combining association rule. In *IEEE International Conference on Web Services (ICWS 20090*, pages 445–452.

Segev, A. and Toch, E. (2011). Context based matching and ranking of Web services for composition. *IEEE Transactions on Services Computing*, 3(2):210–222.

Skoutas, D., Sacharidis, D., Simitsis, A., and Sellis, T. (2010). Ranking and clustering Web services using multicriteria dominance relationships. *IEEE Transactions on Services Computing*, 3(3):163–177.